EIGENCONNECTIONS TO INTRUSION DETECTION

Yacine Bouzida and Sylvain Gombault
Département RSM GET/ ENST Bretagne
2, rue de la Châtaigneraie, CS 17607
35576 Cesson Sévigné CEDEX, FRANCE
{Yacine.Bouzida,Sylvain.Gombault}@enst-bretagne.fr

Abstract Most current intrusion detection systems are signature based ones or machine learning based methods. Despite the number of machine learning algorithms applied to KDD 99 cup, none of them have introduced a pre-model to reduce the huge information quantity present in the different KDD 99 datasets. We introduce a method that applies to the different datasets before performing any of the different machine learning algorithms applied to KDD 99 intrusion detection cup. This method enables us to significantly reduce the information quantity in the different datasets without loss of information. Our method is based on Principal Component Analysis (PCA). It works by projecting data elements onto a feature space, which is actually a vector space R^d, that spans the significant variations among known data elements. We present two well known algorithms we deal with, decision trees and nearest neighbor, and we show the contribution of our approach to alleviate the decision process. We rely on some experiments we perform over network records from the KDD 99 dataset, first by a direct application of these two algorithms on the rough data, second after projection of the different datasets on the new feature space.

Keywords: Intrusion Detection, Principal Component Analysis, KDD 99, Decision Trees, Nearest Neighbor

1. Introduction

A modern computer network should acquire many mechanisms to ensure the security policy of data and equipment inside the network. Intrusion detection systems (IDSs) are an integral package in any well configured and managed computer system or network. IDSs may be some software or hardware systems that monitor the different events occuring in the actual network and analyze them for signs of security threats.

There are two major approaches in intrusion detection: anomaly detection and misuse detection. Misuse detection consists of first recording and repre-

Y. Deswarte et al. (eds.), *Security and Protection in Information Processing Systems*

senting the specific patterns of intrusions that exploit known system vulnerabilities or violate system security policies, then monitoring current applications or network traffic activities for such patterns, and reporting the matches. There are several developed models in misuse intrusion detection [Ilgun, 1993; Kumar and Spafford, 1994]. They differ in representation as well as the matching algorithms employed to detect such threat patterns. Anomaly detection, on the other hand, consists of building models from normal data and then detect variations from the normal model in the observed data. Anomaly detection was originally introduced by Anderson [Anderson, 1980] and Denning [Denning, 1987]. The main advantage with anomaly intrusion algorithms is that they can detect new forms of attacks, because these new intrusions will probably deviate from the normal behavior [Denning, 1987].

There are many IDSs developed during the past three decades. However, most of the commercial and freeware IDS tools are signature based [Roesch, 1999]. Such tools can only detect known attacks previously described by their corresponding signatures. The signature database should be maintained and updated periodically and manually for new attacks. For this reason, many data mining and machine learning algorithms are developed to discover new attacks that are not described in the training labeled data.

Literature survey on intrusion detection indicates that most researchers applied an algorithm directly [Pfahringer, 2000; Agrawal and Joshi, 2000; Levin, 2000] on the rough data obtained from network traffic or other local or remote applications. The majority of the machine learning algorithms applied to anomaly intrusion detection suffers from the high consuming time [Pfahringer, 2000] when applied directly on rough data. The KDD 99 cup intrusion detection datasets [KDD Cup, 1999] are an example where many machine learning algorithms, mostly inductive learning based, were applied directly on the data which is a binary TCPdump data processed into connection records. Each connection record corresponds to a normal connection or to a specified attack as described in section 2.

Much of the previous work on anomaly intrusion detection in general and on the KDD 99 cup datasets in particular ignored the issue of just what measures of the user, application and/or network traffic behavior stimulus are important for intrusion detection. This suggested to us that an information theory approach coding and decoding user/application or connection record behaviors may give new information content of user/attack behaviors, emphasizing the significant local or global *"features"*. These features may or may not be directly related to the actual used metrics or attributes such as CPU consumed time, number of web pages visited during a session in the case of user behaviors and such as the used protocol, service in the case of network connection records. In the remaining of this paper, we will be just interested in network

connection records (for more details on profiles' behaviors, see [Bouzida and Gombault, 2003]).

In the language of information theory, we want to extract the relevant information in a network connection record, encode it efficiently, and compare one network connection record encoding with a database of network connection records encoded similarly. A simple approach to extract the information contained in a network connection record is to capture the variation in a collection of connection records, independently of any judgement of feature, and use this information to encode and compare network connection records.

In mathematical terms, we wish to find the principal components of the distribution of the connection records, or the eigenvectors of the covariance matrix of the set of the connection records [Jolliffe, 2002]. These eigenvectors can be thought of as a set of features which together characterize the variation between records connections. Each connection record location contributes more or less to each eigenvector which we call *"eigenconnection"*. Each connection record can be presented exactly in terms of linear combination of the eigenconnections. Each connection can also be approximated using only the best -*"eigenconnections"*- those that have the largest eigenvalues, and which therefore account for the most variance within the set of connection records. The best N eigenconnections span an N dimensional subconnection -*"connection space"*- of all possible connection records.

This new space is generated by an information theory method called Principal Component Analysis (PCA) [Jolliffe, 2002]. This method has proven to be an exceedingly popular technique for dimensionality reduction and is discussed at length in most texts on multivariate analysis. Its many application areas include data compression [Kirby and Sirovich, 1990], image analysis, visualization, pattern recognition [Turk and Pentland, 1991] and time series prediction.

The most common definition of PCA, due to Hotelling (1933) [Hotelling, 1933], is that, for a set of observed vectors $\{v_i\}$, $i \in \{1, \ldots, N\}$, the q principal axes $\{w_j\}$, $j \in \{1, \ldots, q\}$ are those orthonormal axes onto which the retained variance under projection is maximal. It can be shown that the vectors w_j are given by the q dominent eigenvectors (i.e. those with largest associated eigenvalues) of the covariance matrix $C = \sum_i \frac{(v_i - \overline{v})(v_i - \overline{v})^T}{N}$ such that $Cw_j = \lambda_i w_j$, where $\overline{v}$ is the simple mean. The vector $u_i = W^T(v_i - \overline{v})$, where $W = (w_1, w_2, \ldots, w_q)$, is thus a *q-dimensional* reduced representation of the observed vector v_i.

We investigate, in this paper, an eigenconnection approach based on principal component analysis for anomaly intrusion detection applied to the different KDD 99 intrusion detection cup datasets.

This paper is organized as follows: Section 2 describes the different KDD 99 intrusion detection cup datasets. Sections 3 and 5 introduce the application

of two algorithms; the nearest neighbor and decision trees, where in section 3 these algorithms are briefly presented and in section 5, we present and discuss the different results obtained by using these two algorithms on rough data or after reduction of the feature space using PCA. Section 4 provides the eigen-connection approach for dimensionnality reduction of data. Finally, section 6 concludes the paper.

2. Description of KDD 99 intrusion detection datasets

The main task for the KDD 99 classifier learning contest [KDD Task, 1999] was to provide a predective model able to distinguish between legitimate (normal) and illegitimate (called intrusion or attacks) connections in a computer network. The training dataset contained about 5,000,000 connection records, and the training 10% dataset consisted of 494,021 records among which there were 97,278 normal connections (i.e. 19.69%). Each connection record consists of 41 different attributes that describe the different features of the corresponding connection, and the value of the connection is labeled either as an attack with one specific attack type, or as normal. The 39 different attack types present in the 10% datasets are given in table 1.

Each attack type falls exactly into one of the following four categories:

1 Probing: surveillance and other probing, e.g., port scanning;

2 DOS: denial-of-service, e.g. syn flooding;

3 U2R: unauthorized access to local superuser (root) privileges, e.g., various *"buffer overflow"* attacks;

4 R2L: unauthorized access from a remote machine, e.g. password guessing.

The task was to predict the value of each connection (normal or one of the above attack categories) for each of the connection record of the test dataset containing 311,029 connections.

It is important to note that

- the test data is not from the same probability distribution as the training data and
- the test data includes some specific attack types not in the training data. There are 22 different attacks types out of 39 present in the training dataset. The remaining attacks are present in the test dataset with different rates towards their corresponding categories. There are 4 new U2R attack types in the test dataset that are not present in the training dataset. These new attacks correspond to 92.90% (189/228) of the U2R class in

the test dataset. On the other hand, there are 7 new R2L attack types corresponding to 63% (10196/16189) of the R2L class in the test dataset. In addition, there are only 104 (out of 1126) connection records present in the training dataset corresponding to the known R2L attacks present simultaneously in the two datasets. However, there are 4 new DOS attack types in the test dataset corresponding to 2.85%(6555/229853) of the DOS class in the test dataset and 2 new Probing attacks corresponding to 42.94% (1789/4166) of the Probing class in the test dataset.

Table 1. The different attack types.

Probing	DOS	U2R	R2L
ipsweep, mscan, nmap, portsweep, saint, satan.	apache2, back, land, mailbomb, neptune, pod, processtable, smurf, teardrop, udpstorm.	buffer_overflow, httptunnel, loadmodule, perl, ps, rootkit, sqlattack, xterm.	ftp_write, guess_passwd, imap, multihop, named, phf, sendmail, snmpgetattack, snmpguess, spy, warezclient, warezmaster, worm, xlock, xsnoop.

We ran our experiments using two different machine learning algorithms; the nearest neighbor and decision trees, on the 10 % KDD 99 intrusion detection cup [KDD Cup, 1999] generated by the MIT Lincoln Laboratory. Lincoln Labs set up an environment to acquire nine weeks of raw TCPdump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment, but peppered it with the 39 different attacks types. The TCPdump data collected from the network traffic was transformed into connection records using some data mining techniques [Lee et al., 1999].

3. Nearest neighbor and decision trees

3.1 Nearest Neighbor NN

One of the easiest method in machine learning field is the nearest neighbor method or *NN*. It consists of classifying new observations into their appropriate categories by a simple comparison with the known well classified observations. Recall that the only knowledge we have is a set of x_i, i=1,..,M points correctly classified into categories. It is reasonable to assume that observations which are close together -for some appropriate metric- will have the same classification. Thus, when classifying an unknown sample x, it seems appropriate to weight the evidence of the nearby's heavily. One simple non-parametric decision procedure of this form is the nearest neighbor rule or *NN-rule*. This rule classifies x in the category of its nearest neighbor. More precisely, we call x'

a nearest neighbor to x if $min\ d(x, x_i) = x', where\ i = 1, .., M$ and d is the distance between the two considered points such as the Euclidean distance.

After its first introduction by Fix and Hodges [Fix and Hodges, 1951], the *NN* classifier has been used and improved by many researchers [Bay, 1998; Dasarathy, 1991] and employed on many data sets from UCI repository [Hettich and Bay, 1999]. A common extension is to choose the most common class in the *kNN*. The *kNN* is performed on KDD 99 intrusion detection datasets by Eskin et. al [Eskin et al., 2003]. It was applied for another purpose where the dataset is filtered and the percentage of attacks is reduced to 1.5% in order to perform unsupervised anomaly detection. In the following, we are interested in applying the NN classifier on the different datasets with its simplest form. That is compute all possible distance pairs between all the training data set and the test dataset records.

Since our datasets consist of continuous and discrete attributes values, we have converted the discrete attibutes values to continuous values following the following idea. Consider we have Σ_i possible values for a discrete attribute i. For each discrete attribute correpond $\mid \Sigma_i \mid$ coordinates. There is one coordinate for every possible value of the attribute. Then, the coordinate corresponding to the attribute value has a value of 1 and all other remaining coordinates corresponding to the considered attribute have a value of 0. As an example, if we consider the protocol type attribute which can take one of the following discrete attributes tcp, udp or icmp. Then, there will be three coordinates for this attribute. If the connection record has a tcp (resp. udp or icmp) as a protocol type then the corresponding coordinates will be (1 0 0) (resp. (0 1 0) or (0 0 1)). With this transformation, each connection record in the different KDD 99 datasets will be represented by 125 (3 different values for the *protocol_type*, 11 different values for the *flag* attribute, 67 possible values for the *service attribute* and 0 or 1 for the other remaining 6 discrete attributes) coordinates instead of 41 according to the above discrete attributes values transformation.

3.2 Decision trees

Decision tree induction has been studied in details in both areas of pattern recognition and machine learning. In the vast area concerning decision trees, also known as classification trees or hierarchical classifiers, at least two seminal works are to be mentioned, those by Quinlan [Quinlan, 1986] and those by Breiman et al. [Breiman et al., 1984]. The former synthesizes the experience gained by people working in the area of machine learning and describes a computer program called ID3, which has evolved in a new system, named C4.5 [Quinlan, 1993]. The latter originated in the field of statistical pattern recognition and describes a system, named CART (Classification And Regres-

sion Trees), which has mainly been applied to medical diagnosis. A decision tree is a tree that has three main components: nodes, arcs, and leaves. Each node is labeled with a feature attribute which is most informative among the attributes not yet considered in the path from the root, each arc out of a node is labeled with a feature value for the node's feature and each leaf is labeled with a category or class.

Most of the decision trees algorithms use a top down strategy; i.e from the root to the leaves. Two main processes are necessary to use the decision trees:

- Building process: it consists of building the tree by using the labeled training dataset. An attribute is selected for each node based on how it is more informative than others. Leaves are also assigned to their corresponding class during this process.
- Classification process: A decision tree is important not because it summarizes what we know, i.e. the training set, but because we hope it will classify correctly new cases. Thus, when building classification models, one should have both training data to build the model and test data to verify how well it actually works. New instances are classified by traversing the tree from up to down based on their attribute values and the node values until one leaf is reached that corresponds to the class of the new instance.

We use the C4.5 algorithm [Quinlan, 1993] to construct the decision trees where Shanon Entropy is used to measure how informative is a node. The selection of the best attribute node is based on the gain ratio $GainRatio(S, A)$ where S is a set of records and A a non categorical attribute. This gain defines the expected reduction in entropy due to sorting on A. It is calculated as the following [Mitchell, 1997]:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{\mid S_v \mid}{\mid S \mid} Entropy(S_v) \qquad (1)$$

In general, if we are given a probability distribution $P = (p_1, p_2, .., p_n)$ then the information conveyed by this distribution, which is called the Entropy of P is :

$$Entropy(P) = -\sum_{i=1}^{n} p_i log_2 p_i \qquad (2)$$

If we consider only (1) then an attribute with many values will be automatically selected. One solution is to use $GainRatio$ instead [Quinlan, 1986]

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInformation(S, A)} \qquad (3)$$

where

$$SplitInformation(S, A) = -\sum_{i=1}^{c} \frac{\mid S_i \mid}{\mid S \mid} log_2 \frac{\mid S_i \mid}{\mid S \mid} \tag{4}$$

where S_i is a subset of S for which A has a value v_i.

4. Eigenconnection approach

Principal component analysis (PCA) is a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components. The objective of principal component analysis is to reduce the dimensionality (number of variables) of the dataset but retain most of the original variability in the data. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. In this section, we investigate the eigenconnection approach based on the principal component analysis. In our case, each connection record corresponds to one vector of n variables corresponding to the different attributes in the different datasets. The procedure is the following:

The set of n different measures are collected in a vector called connection record vector representing the corresponding connection. So if Γ is a connection vector then we can write

$$\Gamma = \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{pmatrix} \tag{5}$$

where m_i, $i = 1, \ldots, n$ correspond to the different measures. In most cases, the connection vectors are very similar and they can be described by some *basic connection vectors*.

This approach involves the following initialization procedure:

1 acquire an initial set of connection records (this set is called the training set). In this paper, we use the kdd 99 10% training dataset containing $M = 494,021$ connection records;

2 calculate the eigenconnections from the training set, keeping only n' ($n' \ll n$) eigenconnections that correspond to the highest eigenvalues. These n' connections define the connection space.

3 calculate the corresponding distribution in $n' - dimensional-$ weight space for each known connection record, by projecting their connection vectors onto the *connection space*;

4 (optional) perform a machine learning algorithm (building process) on the new datasets in the new connection space; for the decision tree algorithm, it is necessary to build the tree which will be used in the detection process. However, there is no need to perform the *NN* algorithm at this stage. This is the reason why this step is optional depending on the machine learning algorithm being used.

Having projected the training data onto the new feature space, the following steps are then used to classify and detect intrusions from the new connection records in the test data:

1 calculate a set of weights based on the input connection record and the n' eigenconnections by projecting the input connection record vector onto each eigenconnection,

2 use one of the different machine learning algorithms (classification process) to detect intrusions from the new connection records represented in the new feature space.

4.1 Calculating the eigenconnections

Let the training set of connection vectors be $\Gamma_1, \Gamma_2, \ldots, \Gamma_M$. The average connection Ψ of this set is defined by:

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i \tag{6}$$

Each connection record vector Γ_i differs from the average Ψ by:

$$\Phi_i = \Gamma_i - \Psi \tag{7}$$

The eigenconnections are the eigenvectors of the covariance matrix C where

$$C_{(n \times n)} = \frac{1}{M} \sum_{i=1}^{M} \Phi_i \Phi_i^T = AA^T \tag{8}$$

$$A_{(n \times M)} = \frac{1}{\sqrt{M}} [\Phi_1 \Phi_2 \ldots \Phi_M] \tag{9}$$

Let U_k be the k^{th} eigenvector of C, λ_k the associated eigenvalue and $\mathbf{U}_{(\mathbf{n} \times \mathbf{n}')} = [\mathrm{U}_1 \mathrm{U}_2 \ldots \mathrm{U}_{\mathrm{n}'}]$ the matrix of these eigenvectors (eigenconnections). Then

$$CU_k = \lambda_k U_k \tag{10}$$

such that

$$U_k^T U_l = \begin{cases} 1 & \text{if } k = l \\ 0 & \text{if } k \neq l \end{cases} \tag{11}$$

The feature vector corresponding to the connection record Γ_i is :

$$\Omega_i = U^T \times \Phi_i = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_{n'} \end{pmatrix} \tag{12}$$

If the length of the connection record vector is n (number of considered attributes), the matrix C is $n \times n$. The principal component analysis of the contextual covariance matrix C is obtained by calculating its eigenvalues and eigenvectors, and ordering the eigenvalues (and the corresponding eigenvectors) in decreasing order. The sub-space generated by the eigenvectors corresponding to the highest eigenvalues has the highest inertia. By construction, all the directions of the eigenvectors are orthogonal. The principal components associated with the smallest eigenvalues often correspond to not interesting information [Jolliffe, 2002]. Therefore, they are usually removed. Other strategies can be adopted to select the components [Jolliffe, 2002].

The quantity given by

$$\sum_{i=1}^{n'} \lambda_i \tag{13}$$

is called inertia explained by the subspace generated by the first n' ($n' \ll n$) eigenvectors of C.

In practice, the number of the principal factorial components chosen depends on the precision we wish to reach. In general, we can limit to 2, 3 or 4 considered principal factors (axes). The inertia ratio explained by these axes is

$$\tau = \frac{\sum_{i=1}^{n'} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \tag{14}$$

This ratio defines the information rate kept, from the whole rough input data, by the corresponding n' eigenvalues.

5. Experimental methodology and results

In this section, we will present the different results and experiments obtained when directly applying the two methods discussed in section 3 on the different KDD 99 cup datasets or with a combination with Principal Component Analysis; first by projecting all data on the new space generated by the few number

PCA's principal axes then applying the nearest neighbor or decision trees algorithm on the datasets but after their projection on the new reduced PCA's space.

The accuracy of each experiment is based on the percentage of successful prediction (PSP) on the test dataset.

$$PSP = \frac{number\ of\ successful\ instance\ classification}{number\ of\ instances\ in\ the\ test\ set} \tag{15}$$

5.1 Nearest neighbor with/without PCA

The first experiment, we perform, consists in evaluating the nearest neighbor algorithm on the KDD 99 database. The main problem encountered when computing the nearest neighbor is that it is computationally expensive to compute the nearest neighbor of each point. The complexity of this computation is $O(nm)$ where n is the number of connection records in the training dataset and m in the test dataset. Each distance computation between two connection records depends on the number of space coordinates where they are represented. Of course, this algorithm may be approximated with a cluster based estimation algorithm [Han and Kamber, 2001]. However, the distance between two connection records remains dependent on the coordinates number of the feature space. For this reason, we have projected the different datasets connections on the new feature space generated by the principal component axes. There are 125 coordinates of each connection in KDD 99 after transformation for discrete attribute values as explained in section 3 (41 attributes added to the representation of each discrete value that has at least two coordinates).

Table 2 presents the confusion matrix when applying directly the nearest neighbor on the feature space generated by these 125 coordinates.

Table 2. Confusion matrix obtained with the nearest neighbor algorithm on 125 coordinates.

Predicted as Actual	Normal	Probing	DOS	U2R	R2L
Normal(60593)	**99.50%**	0.26%	0.24%	0.00%	0.00%
Probing(4166)	17.21%	**72.01%**	10.28%	0.00%	0.50%
DOS(229853)	2.87%	0.12%	**97.01%**	0.00%	0.00%
U2R(228)	39.96%	18.80%	32.01%	**6.60%**	2.63%
R2L(16189)	96.12%	2.65%	0.00%	0.02%	**1.21%**
PSP=92.05%					

Using the same algorithm (the nearest neighbor), we have experimented the test dataset on a new feature space generated by at most seven PCA's axes. We

have performed the different experiments by considering 2, 3, ..., or 7 axes. The results are not much different from each other when we consider from 2 to 7 axes. Table 3 shows the confusion matrix when we consider four axes (i.e. each connection record in the different datasets is represented by only four coordinates).

Table 3. Confusion matrix obtained with the nearest neighbor on 4 coordinates after performing PCA .

Predicted as Actual	Normal	Probing	DOS	U2R	R2L
Normal(60593)	**99.50%**	0.27%	0.23%	0.00%	0.00%
Probing(4166)	13.87%	**74.40%**	11.37%	0.00%	0.36%
DOS(229853	2.68%	0.18%	**97.14%**	0.00%	0.00%
U2R(228)	35.96%	14.47%	39.03%	**7.91%**	2.63%
R2L(16189)	97.49%	1.71%	0.00%	0.00%	**0.80%**
PSP=92.22%					

The confusion matrix in table 3 shows that the results after PCA application are slightly better. In addition, the computation time is reduced by a factor of approximately thirty ($\sim 125/4$) when considering 4 principal components. Hence, it is better to reduce the space on which the connection records are represented before applying any machine learning algorithm. This first experimentation is used to show that a combination between PCA and the nearest neighbor performs well even if a few axes are considered (at most seven) to represent the records. According to equation (14), the inertia ratio is close to 1 (0.999) when considering only 4 axes. This is the reason why a representation with only four axes provides a good prediction rate.

In the two experiments, the two last classes R2L and U2R are not well detected. The maximum PSP for U2R class is 7.91% and 1.21% for R2L.

5.2 Decision trees with/without PCA

This section presents experimental results using decision trees with the C4.5 algorithm. This latter is applied, in the first experiment, directly on the different datasets using the whole 41 attributes and then compared to its application on the datasets but after their projection onto the new space generated by the few principal component axes number.

During our experiments, we have considered, as in [BenAmor et al., 2004], two cases. The first consists in grouping the whole 39 attacks types into four attack categories before training. In the second case, they are gathered after classification.

Decision trees without PCA. In this section, we present the different results obtained when applying directly the C4.5 algorithm on rough data. Table 4 evaluates the application of the C4.5 algorithm on the dataset by gathering the whole attacks into four categories before the training step and its application on the rough dataset after classification.

Table 4. Confusion matrix relative to five classes using the C4.5 algorithm. The values between parentheses correspond to gathering the whole attacks results into five categories after classification.

Predicted as Actual	%Normal	%Probing	%DOS	%U2R	%R2L
Normal(60593)	**99.49(99.42)**	0.36(0.39)	0.12(0.15)	0.00(0.00)	0.02(0.03)
Probing (4166)	21.32(15.75)	**74.70(78.80)**	3.98 (5.45)	0.00(0.00)	0.00(0.00)
DOS (229853)	2.68(2.58)	0.00(0.46)	**97.31(96.96)**	0.00(0.00)	0.00(0.00)
U2R (228)	90.79(56.58)	1.75(28.51)	0.44(0.88)	**4.39(5.26)**	2.63(8.77)
R2L (16189)	92.03(94.63)	2.10(0.07)	0.01(0.00)	0.02 (0.03)	**5.84(5.27)**
PSP=92.60%,(PSP=92.35%)					

The different results obtained in this first experiment show that gathering the attacks before training or after classification does not influence the percentage of successful prediction. The two classes U2R and R2L are classified with a percentage of successful prediction of at most 5.84%. This is due to the low number of samples of these two classes in the training set; 0.01% examples of U2R in the training set (resp. 0.23% of R2L) versus 0.07% of U2R (resp. 5.20% of R2L) in the test dataset and to the new forms of these two attacks classes that appear in the test dataset which are not present in the training set.

Decision trees with PCA. We now apply the C4.5 on the new feature space generated by the principal axes. All the training dataset and the test dataset connection records are projected onto the new feature space. This new feature space is generated by at most 7 axes in our different experiments to validate the results of combining PCA with the C4.5 decision trees algorithm.

According to tables 5 and 8 [1], there is a slight difference between the use of decision trees on rough data and their combination with PCA on the new feature space.

However, it is important to mention that the number of nodes in the decision tree generated when we apply C4.5 on rough data is greater than that of nodes in the decision tree when applied with the different datasets but in the new

[1] PCAi corresponds to the projection of the data onto the first i principal axes corresponding to the first i highest eigenvalues.

Table 5. Confusion matrix relative to five classes using the C4.5 algorithm after dataset projection onto two principal component axes. The values between parentheses correspond to gathering the whole attacks results into five categories after classification.

Predicted as Actual	%Normal	%Probing	%DOS	%U2R	%R2L
Normal(60593)	**99.00(98.99)**	0.85(0.84)	0.12(0.12)	0.00(0.00)	0.03(0.04)
Probing(4166)	29.60(30.20)	**66.80(66.30)**	3.50(3.50)	0.10(0.00)	0.00(0.00)
DOS(229853)	2.42(2.42)	0.33 (0.33)	**97.25(97.25)**	0.00 (0.00)	0.00 (0.00)
U2R(228)	92.98(91.23)	0.00(0.00)	0.44(0.00)	**6.58(8.33)**	0.00(0.44)
R2L(16189)	99.94(97.69)	0.00(0.00)	0.06(0.01)	0.00(0.00)	**0.01(2.30)**
PSP=92.05%(PSP=92.16%)					

feature space generated by the PCA. In addition, the training time consumed to construct the decision tree with the new data in the feature space, generated by at most *seven* principal components, is more interesting as presented in table 6.

Table 6. Time and tree size with/without PCA.

	Decision trees without PCA	Decision trees with PCA
Number of nodes	$\sim$ 1500 before pruning $\sim$ 700 after pruning	$\sim$ 330 before pruning $\sim$ 211 after pruning
Training time	$\sim 3mn40sec$	$\sim 50sec$

Furthermore, the problem of the prediction ratio with the last two classes persists always as mentioned in the previous subsection. The highest predicted successful ratio obtained with the R2L class does not exceed 2.30%. This is because the principal components associated with the smallest eigenvalues often correspond to not interesting information [Jolliffe, 2002] that corresponds in reality in our case to the classes that are not present with high rates in the training set and we are taking into account only the highest eigenvalues. This is the reason why the R2L class prediction ratio is very small. To circumvent this problem, we have considered other axes corresponding to lower principal axes. In this case, we have obtained 5.86% as the highest prediction ratio for the R2L class.

Table 7 presents the best prediction ratios when considering other components axes not presented in tables 5 and 8.

According to table 7, the results obtained by combining decision trees with PCA are slightly better than those in table 4 when applying directly decision trees on rough data. We may improve the prediction ratio, when combining

Table 7. The best prediction ratios obtained for each class when considering different principal components.

Attack Category	Normal (60593)	Probing (4166)	DOS (229853)	U2R (228)	R2L (16189)
Detection Ratio	99.52%	78.84%	98.26%	12.72%	5.86%
PSP=92.63%					

decision trees with PCA, of the last two classes by duplicating their different samples in the training set. By this reasonning, they will not be considered as less interesting information.

6. Conclusion

We have presented in this paper a new idea on how to reduce the different representation spaces before applying some machine learning algorithms on the different KDD 99 intrusion detection datasets. This new representation permits to improve the learning time and space representation of the different datasets with a similar successful prediction in the whole experiments.

The main drawback which persists in combining decision trees or the nearest neighbor with PCA is the poor prediction ratio rate of the R2L class which is in most of the time classified as normal. This is due to its low presence in the training dataset (0.23%). We may improve this ratio by boosting the number of samples of this class in the training dataset before applying the PCA algorithm in order to transform it into an interesting information class represented by a principal component corresponding to a higher eigenvalue.

However, the new attacks types that are present in the two last classes, R2L and U2R, in the test data set could not be detected by machine learning algorithms [Sabhnani and Serpen, 2004]. This suggests to perform other unsupervised machine learning or data mining algorithms to deal with these new attacks that should be detected as new attacks. Hence, we may add a new class that we call new attacks class to which new attacks, which are not looking similar to the known attacks in the training dataset, will be classified. This will be discussed in a forthcoming paper.

Table 8. Confusion matrix relative to five classes using the C4.5 algorithm after dataset projection onto 3,4,5,6 or 7 principal component axes. The values between parentheses correspond to gathering the whole attacks results into five categories after classification.

Predicted as Actual	%Normal	%Probing	%DOS	%U2R	%R2L
Normal	**(60593)**				
PCA3	**99.47(99.52)**	0.35(0.32)	0.13(0.12)	0.00(0.00)	0.05(0.04)
PCA4	**99.43(99.51)**	0.36(0.34)	0.13(0.12)	0.01(0.01)	0.07(0.02)
PCA5	**99.23(99.19)**	0.39(0.37)	0.22(0.27)	0.00(0.16)	0.15(0.02)
PCA6	**99.46(99.47)**	0.36(0.34)	0.16(0.15)	0.00(0.02)	0.02(0.02)
PCA7	**99.39(99.42)**	0.35(0.14)	0.24(0.02)	0.00(0.06)	0.02(0.36)
Probing		**(4166)**			
PCA3	28.88(30.63)	**67.67(62.12)**	3.36(7.25)	0.10(0.00)	0.00(0.00)
PCA4	24.20(30.56)	**69.40(63.08)**	6.39(6.34)	0.00(0.00)	0.02(0.02)
PCA5	13.85(16.30)	**76.26(73.09)**	8.93(10.30)	0.00(0.29)	0.96(0.02)
PCA6	18.31(22.56)	**76.48(69.40)**	5.21(8.02)	0.00(0.02)	0.00(0.00)
PCA7	14.98(22.71)	**75.76(69.28)**	9.27(8.02)	0.00(0.00)	0.00(0.00)
DOS			**(229853)**		
PCA3	2.60(2.75)	0.20(0.00)	**97.18(97.25)**	0.00(0.00)	0.02(0.00)
PCA4	2.53(2.76)	0.27(0.05)	**97.17(97.17)**	0.00(0.00)	0.02(0.02)
PCA5	2.70(2.69)	0.02(0.06)	**97.26(97.16)**	0.00(0.00)	0.02(0.09)
PCA6	2.76(2.77)	0.02(0.03)	**97.20(97.06)**	0.00(0.00)	0.02(0.14)
PCA7	2.71(2.75)	0.04(0.03)	**97.22(97.06)**	0.00(0.00)	0.02(0.16)
U2R				**(228)**	
PCA3	85.09(65.35)	1.75(9.65)	0.44(14.91)	**12.72(8.33)**	0.00(1.75)
PCA4	68.86(77.63)	9.65(5.70)	9.65(9.65)	**5.26(6.58)**	6.58(0.44)
PCA5	35.53(62.28)	47.37(19.74)	11.40(13.60)	**4.39(3.95)**	1.32(0.44)
PCA6	38.16(41.23)	47.37(3.51)	7.46(48.25)	**5.70(4.39)**	1.32(2.63)
PCA7	38.16(41.23)	47.37(3.51)	7.46(48.25)	**5.70(4.39)**	1.32(2.63)
R2L					**(16189)**
PCA3	99.60(99.60)	0.02(0.02)	0.02(0.02)	0.01(0.01)	**0.35(0.35)**
PCA4	99.83(99.81)	0.03(0.01)	0.03(0.03)	0.01(0.02)	**0.11(0.12)**
PCA5	99.81(99.38)	0.02(0.14)	0.04(0.04)	0.03(0.04)	**0.10(0.41)**
PCA6	99.82(99.43)	0.02(0.14)	0.03(0.02)	0.04(0.05)	**0.09(0.36)**
PCA7	99.81(99.42)	0.02(0.14)	0.02(0.02)	0.06(0.06)	**0.09(0.36)**
PSP3=92.12%(PSP3=92.11%),PSP4=92.12%(PSP4=92.05%) PSP5=92.23%(PSP5=92.13%),PSP6=92.24%(PSP6=92.06%) PSP7=92.23%(PSP7=92.05%)					

Acknowledgements

The authors thank Nora and Frédéric Cuppens for the different discussions and their helpful comments on early versions of this paper.

References

Agrawal, R. and Joshi, M. V. (2000). PNrule: A New Framework for Learning Classifier Models in Data Mining A Case-Study in Network Intrusion detection. Technical Report RC-21719, IBM Research Division.

Anderson, J. P. (1980). Computer Security Threat Monitoring and Surveillance. Technical report, James. P. Anderson Co., Fort Washington, Pennsylvania.

Bay, S. D. (1998). Combining Nearest Neighbor Classifiers Through Multiple Feature Subsets. In *Proceedings of the 15th International Conf. on Machine Learning*, pages 37–45, San Francisco, CA. Morgan Kaufmann.

BenAmor, N., Benferhat, S., and ElOuedi, Z. (2004). Naive Bayes vs Decision Trees in Intrusion Detection Systems. In *The 19th ACM Symposium On Applied Computing - SAC 2004*, Nicosia, Cyprus.

Bouzida, Y. and Gombault, S. (2003). Intrusion Detection Using Principal Component Analysis. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). Classification and Regression Trees.

Dasarathy, B. V. (1991). A Computational Demand Optimization Aide for Nearest-Neighbor-Based Decision systems. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1777–1782. Morgan Kaufmann.

Denning, D. (1987). An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232.

Eskin, E., Arnold, A., Prerau, M., Portnoy, L., and Stolfo, S. (2003). A Geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Applications of Data Mining in Computer Security*.

Fix, E. and Hodges, J. L. (1951). Discriminatory analysis: Nonparametric discrimination: Consistency properties. Technical Report 21-49-004, USAF School of Aviation Medecine, Randolf Field, Texas.

Han, J. and Kamber, M. (2001). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers.

Hettich, S. and Bay, S. D. (1999). The UCI KDD Archive. Available at: http://kdd.ics.uci.edu/.

Hotelling, H. (1933). Analysis of a complex statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441.

Ilgun, K. (1993). Ustat, a real time intrusion detection system for UNIX. In *IEEE Symposium on Security and Privacy*, pages 16–28, Oakland, CA.

Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer Verlag, New York, NY, third edition.

KDD Cup (1999). KDD Cup 99 Intrusion Detection Datasets. Available at: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

KDD Task (1999). KDD 99 Task. Available at: http://kdd.ics.uci.edu/databases/kddcup99/task.html.

Kirby, M. and Sirovich, L. (1990). Application of the KarhunenLoeve Procedure for the Characterization of Human Faces. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, 12(1):103–107.

Kumar, S. and Spafford, E. (1994). A pattern matching model for misuse intrusion detection. In *Proceedings of the 17th National Computer security Conference*, pages 11–21.

Lee, W., Stolfo, S. J., and Mok, K. (1999). Mining in a data flow environment: Experience in intrusion detection. In *Proceeding of the 1999 Conference on Knowledge Discovery and Data Mining KDD-99*.

Levin, I. (2000). KDD-99 Classifier Learning Contest LLSoft's Results Overview. SIGKDD Explorations. *ACM SIGKDD*, 1:67–71.

Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.

Pfahringer, B. (2000). Winning the KDD Classification Cup: Bagged Boosting. SIGKDD Explorations. *ACM SIGKDD*, 1:65–66.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:1–106.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers.

Roesch, M. (1999). Snort - Lightweight Intrusion Detection for Networks. In *13th Systems Administration Conference - LISA 99*.

Sabhnani, M. and Serpen, G. (2004). On Failure of Machine Learning Algorithms for detecting Misuse in KDD intrusion Detection Data Set. *Intelligent Analysis*. To Appear.

Turk, M. and Pentland, A. (1991). Eigenfaces for Recognition. *Cognitive Neuroscience*, 13(1):71–96.